MULTIPLE LINEAR REGRESSION

Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. It is an important regression algorithm that models the linear relationship between a single dependent continuous variable and more than one independent variable. It uses two or more independent variables to predict a dependent variable by fitting a best linear relationship.

Why do we use multiple linear regression?

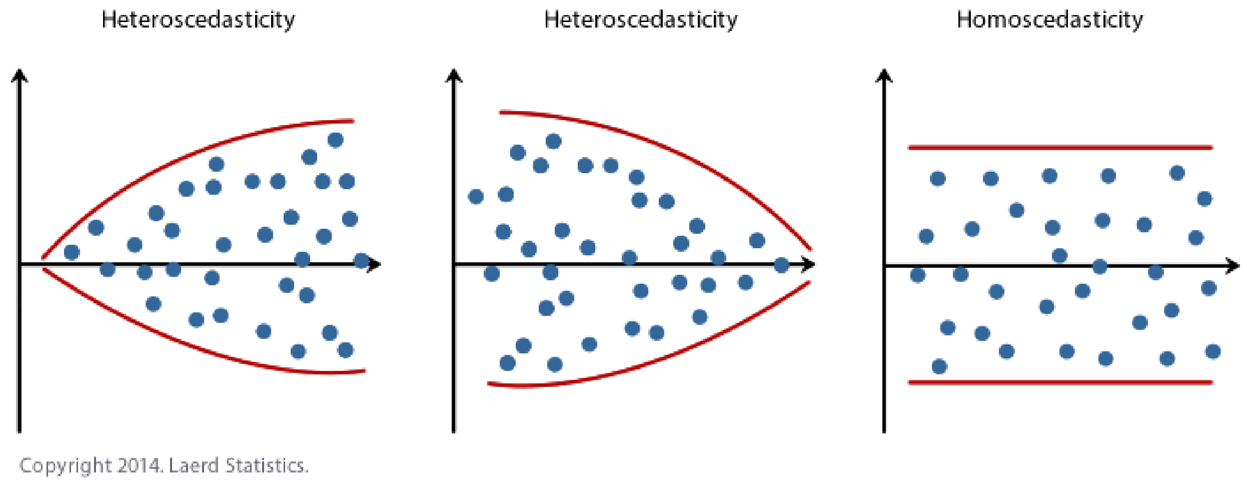
Multiple linear regression is the most common and most important form of regression analysis and is used to predict the outcome of a variable based on two or more independent (or explanatory) variables. Why? By using multiple regression, we can see what the relative influence of one variable is on another. For instance, we can see if the number of bathrooms in a house affects price, holding all else equal. If we only have one variable affecting our dependent variable (housing prices), we would be creating a simple linear regression model. But what if we believed that more than one thing affects housing prices (i.e. view, neighborhood, location to nearest city).

This is where multiple linear regression becomes a great tool. Multiple regression can take many independent variables into a model to explain how those independent variables affect the dependent variable. In our example above, if we wanted a model that explains our dependent variable more succinctly, we could add variables like school-district, crime rate, or another variable that may help predict your response variable (price).

Before we dive into our analysis of multiple linear regression, we must understand what we assume about our data when creating any regression model, and what our variables mean. Then we'll explore how to understand and interpret our model.

Assumptions of multiple linear regression:

1. **Homogeneity of variance or homoscedasticity:** this assumes that the size of the error in our prediction doesn't change significantly across the values of the independent variable.

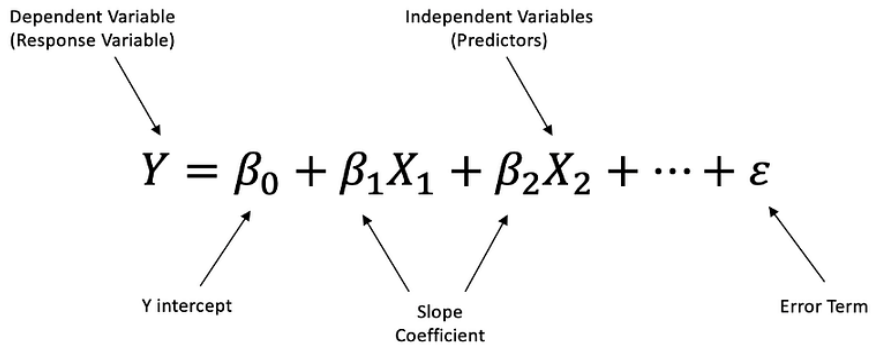


2. Independence of observations — This assumption states that the observations within our data are independent of one another and have been collected through statistically valid methods. **Multicollinearity**: In multiple linear regression, it is possible that some of your explanatory variables are correlated with one another. This is known as **multicollinearity.** For example, height and weight are heavily correlated. If both these variables are used to predict sex, we should only use one of these independent variables in our model as this can create redundant information and would skew the results in our regression model.

3. Normality: We assume our data is normally distributed.

4. Linearity: Multiple linear regression requires the relationship between the independent and dependent variables to be linear.

Understanding the variables in your equation:

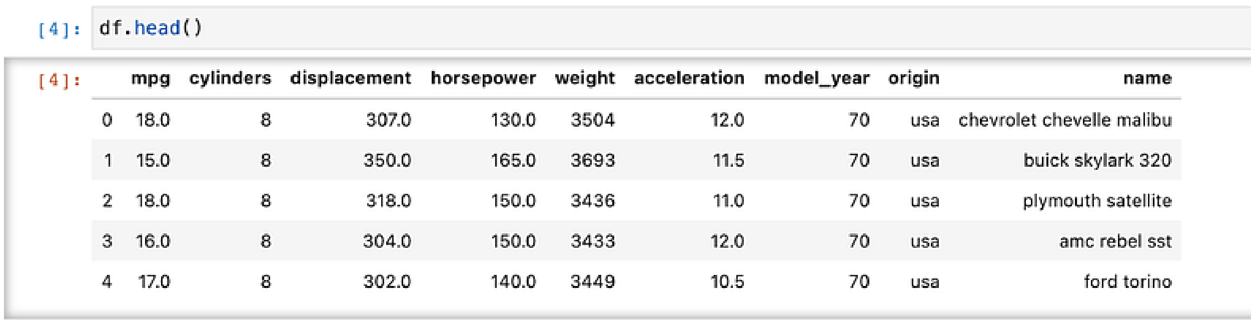

Dependent Variable (Response Variable)
Independent Variables (Predictors)
$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \varepsilon$
Y intercept
Slope Coefficient
Error Term

Below is a list of what each variable represents, as seen in the picture above:

- Y = the dependent or response variable. This is the variable you are looking to predict.
- B0 = This is the y-intercept, which is the value of y when all other parameters are set to 0 (independent variables and error term).
- B1X1= (B1) is the coefficient of the first independent variable (X1) in your model. This can be interpreted as the effect that changing the value of the independent variable has on the predicted y value, holding all else equal.
- "…" = the additional variables you have in your model.
- e = this is the model error. It explains how much variation there is in our prediction of y.

Understanding and interpreting our model:

Below, we have some data on miles per gallon for various cars. Each variable represents a different aspect of the car (i.e. horsepower, weight, acceleration). For example, the Ford Torino has a mpg of 17.0 with a weight of 3,449 pounds and was built in the United States. It is very important for us to understand our data and what it is saying before we create our model.



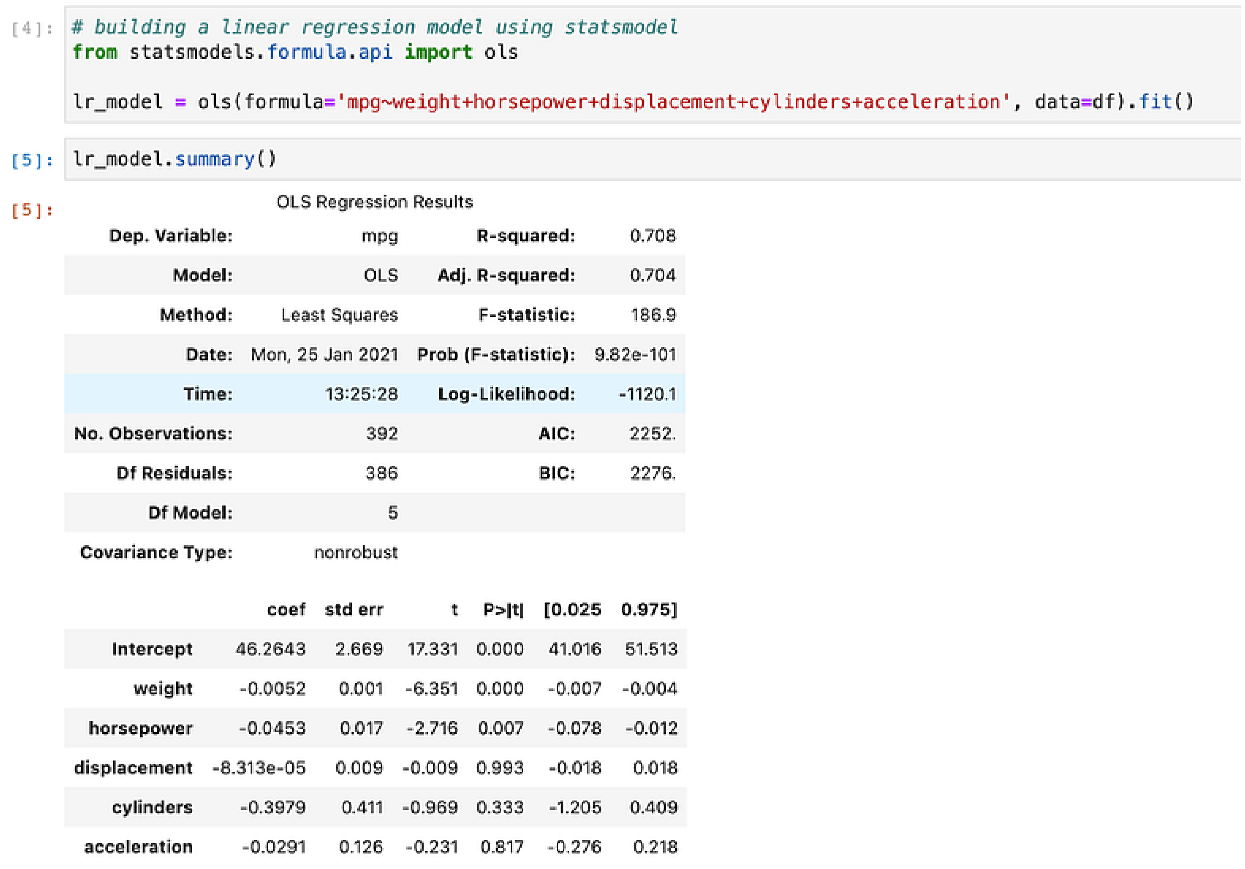
```
[4]: df.head()
```

[4]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

Now that we have familiarized ourselves with our data, let's identify what we are looking to predict (our Y or response variable). For example, a client of yours might want to be able to predict the mpg of a car based on certain parameters. In this case, our mpg variable will be our dependent variable, and we will use other variables in our data set to help predict mpg.

Once we have identified our response variable, we now must decide which independent variables will be the best predictors of our model. I begin by creating a model with each quantitative variable in our data set. Ultimately, we want to be able to understand what variables influence our dependent variable at a statistically significant level. There will be three key factors to help us determine if we keep or remove the variable in question (adjusted r-squared, t critical value, and p-value).



```
[4]: # building a linear regression model using statsmodel
     from statsmodels.formula.api import ols

     lr_model = ols(formula='mpg~weight+horsepower+displacement+cylinders+acceleration', data=df).fit()
```

```
[5]: lr_model.summary()
```

[5]:

OLS Regression Results

Dep. Variable:	mpg	**R-squared:**	0.708
Model:	OLS	**Adj. R-squared:**	0.704
Method:	Least Squares	**F-statistic:**	186.9
Date:	Mon, 25 Jan 2021	**Prob (F-statistic):**	9.82e-101
Time:	13:25:28	**Log-Likelihood:**	-1120.1
No. Observations:	392	**AIC:**	2252.
Df Residuals:	386	**BIC:**	2276.
Df Model:	5		
Covariance Type:	nonrobust		

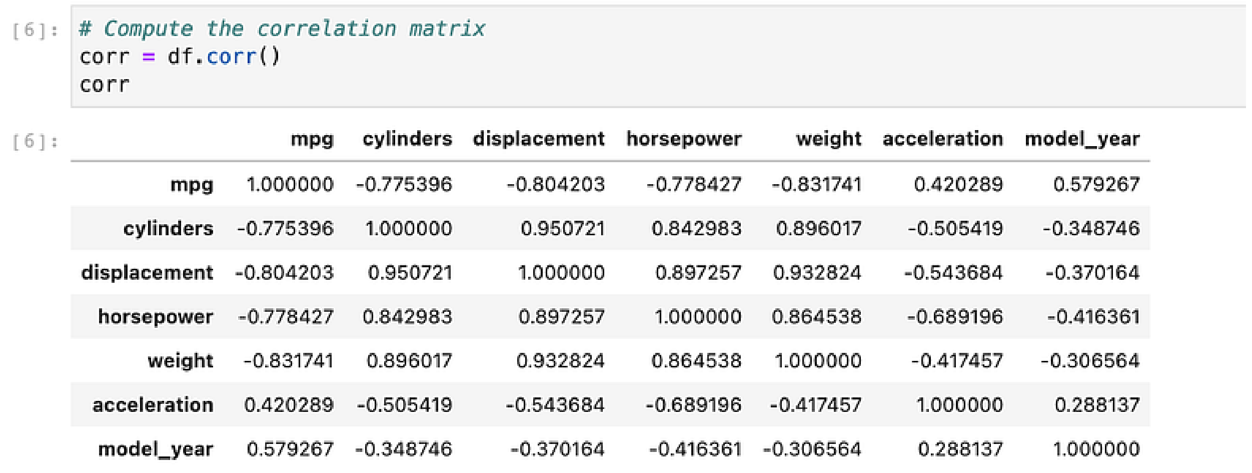
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	46.2643	2.669	17.331	0.000	41.016	51.513
weight	-0.0052	0.001	-6.351	0.000	-0.007	-0.004
horsepower	-0.0453	0.017	-2.716	0.007	-0.078	-0.012
displacement	-8.313e-05	0.009	-0.009	0.993	-0.018	0.018
cylinders	-0.3979	0.411	-0.969	0.333	-1.205	0.409
acceleration	-0.0291	0.126	-0.231	0.817	-0.276	0.218

To begin, let's take a look at our coefficients to understand what they are saying. For example, every additional pound of weight a car has, holding all else equal, brings down mpg by 0.0052. A better way to interpret this is to put it in more relevant and understandable terms; as weight goes up 100 pounds, mpg goes down by 0.52. Similarly, one additional cylinder brings down mpg by 0.3979. Understanding your coefficients is very important in conveying the results of your model.

One way of understanding whether a variable should be kept in your model is by looking at the t-stat (t) value and its associated p-value (P>|t|). The p-value represents how likely it is that the t-statistic would have occurred if the null hypothesis (no relationship between independent variable and dependent variable) between the independent and dependent variables was true. If the p-value for a coefficient is below the alpha (usually set at 0.05) we can conclude that the given independent variable likely influences the dependent variable. For our purposes, both weight and horsepower seem to influence mpg.

What if our variables are correlated? Can we keep those in the model? The simple answer is no. As we stated below, an assumption (and goal) of regression analysis is to have each relationship between the independent variables and the dependent variable be **independent**. The idea is that you can alter the value of one independent variable and keep the other variables constant. If independent variables are in fact correlated, this indicates that shifts in one variable are associated with changes in the other variable. The higher/stronger the correlation between those two independent variables, the more difficult it becomes for the model to estimate the relationship between each explanatory variable and the response variable, due to the fact that the independent variables change in unison.

In the table below, we can see the correlation between each variable:



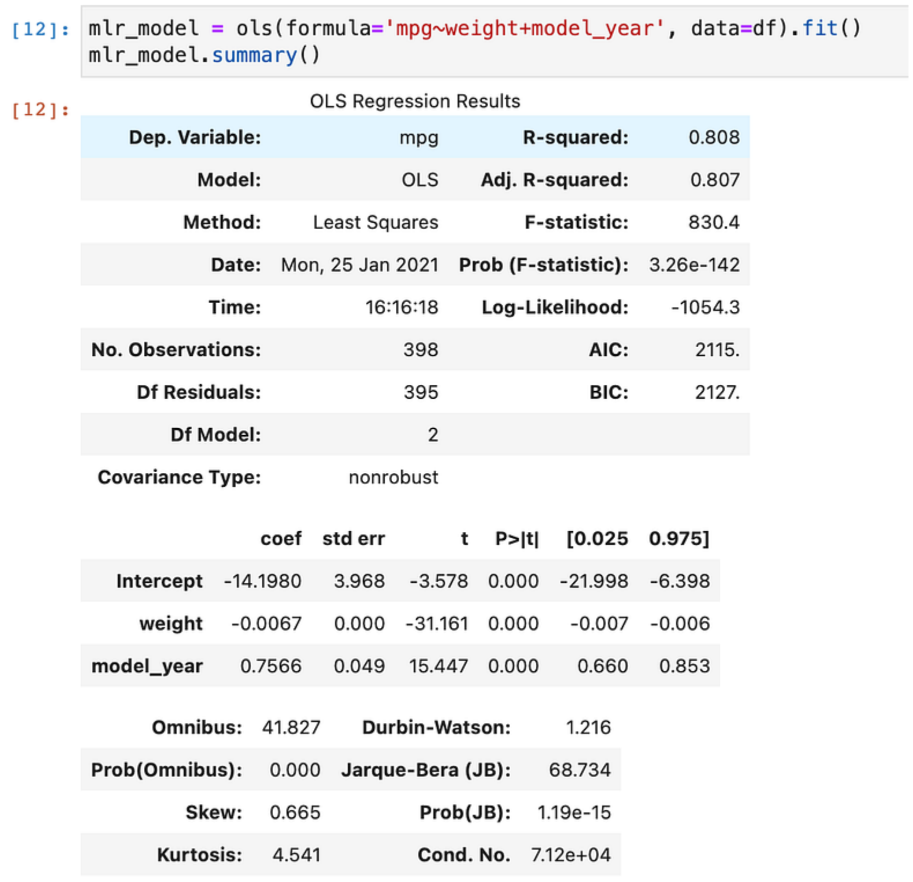
```
[6]: # Compute the correlation matrix
     corr = df.corr()
     corr
```

[6]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

For the purposes of our model, we will use 0.8 as the number that indicates multicollinearity. If the correlation between two variables is above that threshold, we will remove one of the two variables from our model. As you can see, weight and horsepower are heavily correlated, so we will have to take one of those out of our model. I chose to remove horsepower as I believe weight is a stronger indicator of mpg than horsepower. Our acceleration and model_year variables are not correlated with any other variables in our data frame, so we add them to the model as well. However, the coefficient for acceleration has a value over 0.05, so we decide to take that variable out of the model. In our final model, we are left with only weight and model_year as our explanatory variables for mpg.

Notice that our adjusted r-squared is significantly higher in this model than the previous one with all of our variables. Our adjusted r-squared is stating that 80.8% of our variation in the data can be explained by our model which is greater than our previous adjusted r-squared value of 70.4%.



```
[12]: mlr_model = ols(formula='mpg~weight+model_year', data=df).fit()
      mlr_model.summary()
```

[12]:

OLS Regression Results

Dep. Variable:	mpg	**R-squared:**	0.808
Model:	OLS	**Adj. R-squared:**	0.807
Method:	Least Squares	**F-statistic:**	830.4
Date:	Mon, 25 Jan 2021	**Prob (F-statistic):**	3.26e-142
Time:	16:16:18	**Log-Likelihood:**	-1054.3
No. Observations:	398	**AIC:**	2115.
Df Residuals:	395	**BIC:**	2127.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-14.1980	3.968	-3.578	0.000	-21.998	-6.398
weight	-0.0067	0.000	-31.161	0.000	-0.007	-0.006
model_year	0.7566	0.049	15.447	0.000	0.660	0.853

Omnibus:	41.827	**Durbin-Watson:**	1.216
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	68.734
Skew:	0.665	**Prob(JB):**	1.19e-15
Kurtosis:	4.541	**Cond. No.**	7.12e+04

Implement Multiple Linear Regression in Python

In this example, we will use the startup data concerning the profit of startup companies based on several factors. In this dataset, we have columns regarding investments in *R&D, Administration, Marketing* and their location *State*, and the *Profit* dependent on them.

Step 1: Import the required python packages

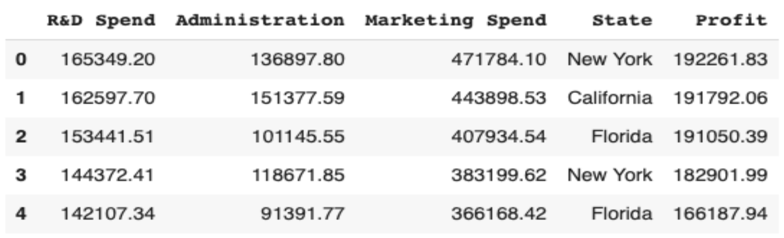
We need *Pandas* for data manipulation, *NumPy* for mathematical calculations, *MatplotLib, and Seaborn* for visualizations. *Sklearn* libraries are used for machine learning operations

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
```

Step 2: Load the dataset

Download the dataset and upload it to your notebook and read it into the pandas dataframe.

```
# Get dataset
df_start = pd.read_csv('/content/50_Startups.csv')
df_start.head()
```

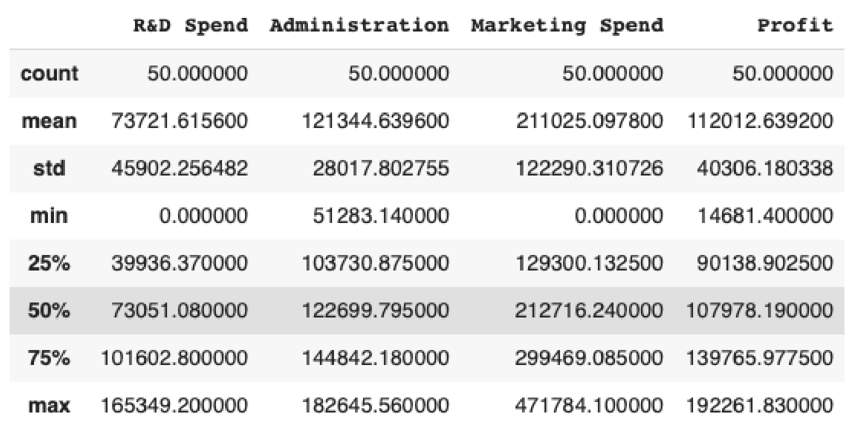


	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

Step 3: Data analysis

Now that we have our data ready, let's analyze and understand its trend in detail. To do that we can first describe the data below -

```
# Describe data
df_start.describe()
```

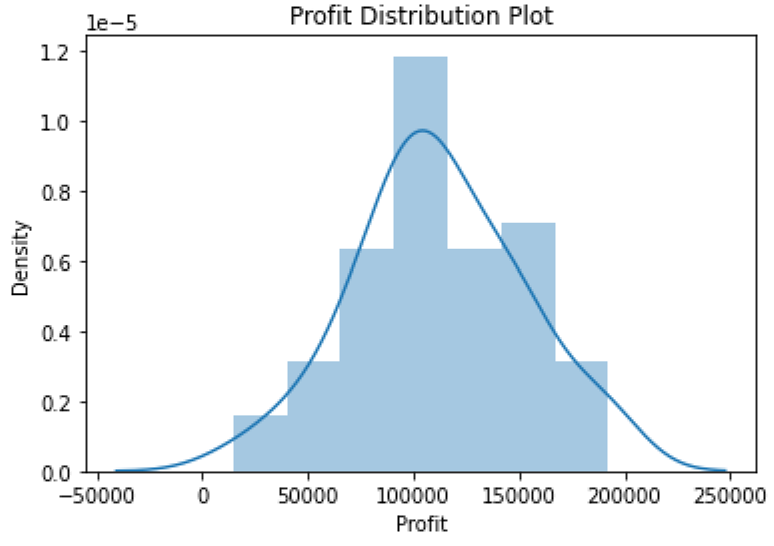


	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

Here, we can see *Profit* ranges from 14681 to 192261 and a median of 107978.

We can also find how the data is distributed visually using *Seaborn* distplot

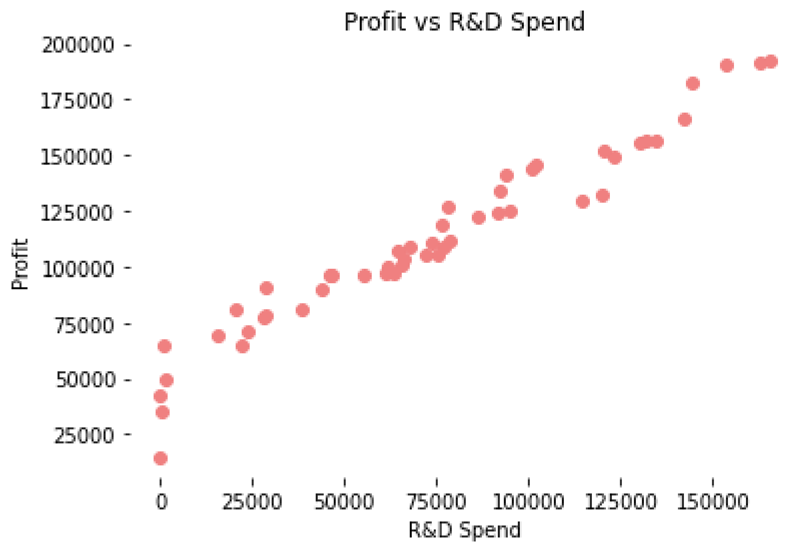
```
# Data distribution
plt.title('Profit Distribution Plot')
sns.distplot(df_start['Profit'])
plt.show()
```



A distplot or distribution plot shows the variation in the data distribution.
It represents the data by combining a line with a histogram.

We can also check the relationship between *Profit* and *R&D Spend* to check data linearity.

```
# Relationship between Profit and R&D Spend
plt.scatter(df_start['R&D Spend'], df_start['Profit'], color = 'lightcoral')
plt.title('Profit vs R&D Spend')
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
plt.box(False)
plt.show()
```



Step 4: Split the dataset into dependent/independent variables

Experience ***(X)*** is the independent variable
Salary ***(y)*** is dependent on experience

```
# Split dataset in dependent/independent variables
X = df_start.iloc[:, :-1].values
y = df_start.iloc[:, -1].values
```

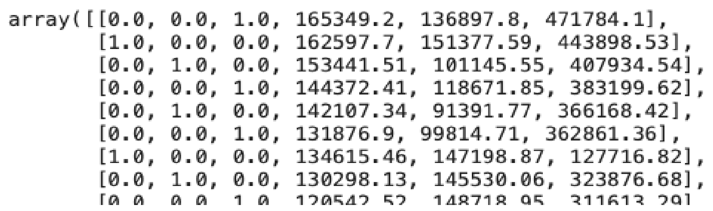
Step 5: One-Hot Encoding of categorical data

Categorical data are variables that contain label values rather than numeric values.

One-Hot Encoding a method to represent a categorical variable in a numerical way, by creating new binary columns (also known as "dummy variables") for each category in the original variable. Each row in the dataset will have a value of 1 in only one of the new columns, indicating the presence of that category, and a value of 0 in all the other columns. This method allows categorical variables to be used in machine learning models that expect numerical inputs. It's called "one-hot" because each category value is represented as a single 1 in a unique column, with all other values in that row being 0.

Here we will encode the *State* column[3] into its binary representation.

```
# One-hot encoding of categorical data
ct = ColumnTransformer(transformers = [('encoder', OneHotEncoder(), [3])], remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
```



```
array([[0.0, 0.0, 1.0, 165349.2, 136897.8, 471784.1],
       [1.0, 0.0, 0.0, 162597.7, 151377.59, 443898.53],
       [0.0, 1.0, 0.0, 153441.51, 101145.55, 407934.54],
       [0.0, 0.0, 1.0, 144372.41, 118671.85, 383199.62],
       [0.0, 1.0, 0.0, 142107.34, 91391.77, 366168.42],
       [0.0, 0.0, 1.0, 131876.9, 99814.71, 362861.36],
       [1.0, 0.0, 0.0, 134615.46, 147198.87, 127716.82],
       [0.0, 1.0, 0.0, 130298.13, 145530.06, 323876.68],
       [0.0, 0.0, 1.0, 120542.52, 148718.95, 311613.29]
```

Step 6: Split data into Train/Test sets

Further, split your data into training (80%) and test (20%) sets using ***train_test_split***

```
# Splitting dataset into test/train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Step 7: Train the regression model

Pass the *X_train* and *y_train* data into the regressor model by ***regressor.fit*** to train the model with our training data.

```
# Regressor model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Step 8: Predict the result

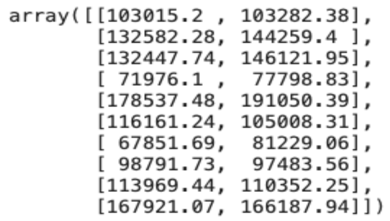
Here comes the interesting part, when we are all set and ready to predict any value of ***y*** *(Profit)* dependent on ***X*** *(R&D Spend, Administration, Marketing Spend)* with the trained model using ***regressor.predict***

```
# Predict result
y_pred = regressor.predict(X_test)
```

Compare results

As we have multiple independent variables, we cannot plot the graph to compare results visually, instead, we can compare a few records of predicted results and actual values side by side.

```
# Compare predicted result with actual value
np.set_printoptions(precision = 2)
result = np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape(len(y_test), 1)), 1)
result
```

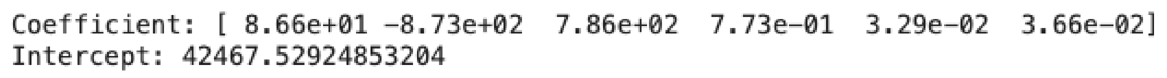


```
array([[103015.2 , 103282.38],
       [132582.28, 144259.4 ],
       [132447.74, 146121.95],
       [ 71976.1 ,  77798.83],
       [178537.48, 191050.39],
       [116161.24, 105008.31],
       [ 67851.69,  81229.06],
       [ 98791.73,  97483.56],
       [113969.44, 110352.25],
       [167921.07, 166187.94]])
```

Coefficients

We can get the coefficients and intercept of multiple linear regression equation $y = \beta 0 + \beta 1x1 + \beta 2x2 + \beta 3x3 + \dots + \beta nxn$ from the regressor model.

```
# Regressor coefficients and intercept
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
```



```
Coefficient: [ 8.66e+01 -8.73e+02  7.86e+02  7.73e-01  3.29e-02  3.66e-02]
Intercept: 42467.52924853204
```

Once the coefficients are estimated, the multiple linear regression model can be used to make predictions for new data. The quality of the predictions can be evaluated using various measures such as ***Mean Squared Error, R-Squared, adjusted R-Squared***, and others.